爱创课堂前端培训

CSS

第 2 天课堂笔记（本课程共 8 天）

班级：北京前端训练营 7 期

日期：2017 年 5 月 11 日

爱创课堂官网 ： www.icketang.com

目录

复习

js 书写位置：

body 标签的最底部。

实际工作中使用书写在 head 标签内一对 script 标签里。

alert()弹出框。

console.log()

字面量：

数字字面量：整数，浮点数，特殊字符（Infinity,NaN）。

字符串字面量：所有的语言， 语言中的数字，特殊字符串。

变量：

定义变量

标识符的规则：第一个字符是字母（a-z,A-Z），下划线，$。其他字符可以是字母（a-z,A-Z），下划线，$，数字。（还不能使用 js 规定的关键字，保留字）。

```
var a;
```

变量赋值：是把右边的值赋值给左边。

```
a = 10;
```

通常我们将变量声明和赋值写一起：

```
var a = 10;
```

变量声明提升：只提升声明，不提升变量的赋值。

数据类型：typeof()

number:数字类型。整数，浮点数，特殊字符。

string：字符串类型。

undefined:undefined 类型。表示未定义。自己就是一个类型。

boolean:布尔类型。true，false。

null:也是自己是一个类型。

数字类型转换为字符串：数字+“”；

```
console.log(12 + “3”);    //123
```

字符串类型转换为数字：

parseInt()：转换为整数。

```
console.log(parseInt(“12.234 你好世界.233”));        //12
```

parseFloat():将字符串转换为浮点数。

```
console.log(parseFloat(“78.2341e3.7898 世界”));    //78234.1
```

一、运算符

运算符（Operators，也翻译为操作符），是发起运算的最简单形式。

分类：（运算符的分类仁者见智，本课程进行一下分类。）

数学运算符(Arithmetic operators)

比较运算符(Comparison operators)

逻辑运算符(Logical operators)

赋值运算符(Assignment operators)

按位运算符(Bitwise operators)

条件（三元）运算符(Conditional operator)

1.1 数学运算符

数学运算符：+,-,*,/,%,()。

运算顺序：先算乘除取余，再算加减。有小括号的先算小括号里面的。

```
// %取余操作，就是要余数部分。
// 余数的范围小于除数
// 范围是[0,除数-1];
console.log(12 % 5);
```

通常我们说的数学运算指的是纯数字和纯数字之间的运算。

①纯数字的字符串和纯数字进行数学运算时，除加法外其他的都会进行隐式转换。

```
console.log("12" - 2);
console.log("12" * 2);
console.log("12" / 2);
console.log("12" % 2);
```

```
10
24
6
0
```

②特殊字符布尔和 null 在与数字进行数学运算时，也进行隐式转换。

true→1,false→0,null→0

```
console.log(5 * true);
console.log(5 * false);
console.log(5 * null);
```

```
5
0
0
```

加法运算也会进行隐式转换：

```
console.log(5 + true);
console.log(5 + false);
console.log(5 + null);
```

```
6
5
5
```

③undefined 和其他字符串与数字进行数学运算时（除加法外），得到的都是 NaN。

```
console.log(5 * undefined);
console.log(5 * "hello");
console.log(5 * "你好");
```

```
NaN
NaN
NaN
```

```
1  console.log(5 + undefined);
2  console.log(5 + "hello");
3  console.log(5 + "你好");
```

```
NaN
5hello
5你好
```

④NaN 和数字进行数学运算时得到的是 NaN 。

```
1  console.log(12 + NaN);
2  console.log(12 - NaN);
3  console.log(12 * NaN);
4  console.log(12 / NaN);
5  console.log(12 % NaN);
```

```
NaN
NaN
NaN
NaN
NaN
```

⑤Infinity 进行数学运算时。

```
1  console.log(12 + Infinity);
2  console.log(12 - Infinity);
3  console.log(12 * Infinity);
4  console.log(12 / Infinity);
5  console.log(12 % Infinity);
```

```
Infinity
-Infinity
Infinity
0
12
```

小测试：

计算下列算式，并将结果输出：

$$\frac{324(23+214)}{568-129} - 11(235-24)$$

1.2 Math 对象

Math 对象是 js 内置的功能非常强大的数学对象。包含数学中所有的属性和方法。

Math.random() 随机数，随机出现一个大于等于 0 小于 1 的一个数。[0,1)。

Math.pow(num,power)。幂的计算。有两个参数，一个参数是书写的底数，第二个参数表示幂。

Math.sqrt(num)。开方的计算。

Math.PI 得到的是 π 的数值。

```
// 随机数，[0,1)
        console.log(Math.random());
        // 幂的计算
        console.log(Math.pow(3,4));
        console.log(Math.pow(780,7));
        // 开方的计算
        console.log(Math.sqrt(81));
        // 得到π的数值
        console.log(Math.PI);
```

```
0.5359296888382308
81
17565568549120000000
9
3.141592653589793
```

1.3 比较运算符

比较运算符：

>	大于
<	小于
==	等于
===	全等于
>=	大于等于
<=	小于等于
!=	不等于
!==	不全等于

不管是纯数字的比较或者是纯数字和字符串的比较等有比较运算符参与的计算得到的结果都是布尔值（true，false）。

```
console.log(7 >= 8);
console.log(7 <= 8);
```

```
false
true
```

特殊情况：

①纯数字字符串与数字进行比较。会进行隐式转换。

```
console.log("12" < 8);
false
undefined
console.log("12" > 8)
true
```

②true，false，null 进行比较比较运算时，进行隐式转换，true→1，false→0，null→0。但是 null 进行==或者===运算时，不等于 0.

③==，等于。==在判断等号两边关系时，会尽可能让两边相等。

```
// ==会尽量让两边关系相等。
console.log(true == 1);
console.log(false == 0);
console.log(null == 0);
```

```
true
true
false
true
true
```

null == 0输出false。特殊情况单独记忆

④===全等于，===在判断两边关系时，会让两边尽可能不等。不但会判断数值相等，还会判断数据类型相同。

```
// ===会尽量让两边不等
console.log("12" === 12);
console.log(12 === 12);
console.log(true === 1);
console.log(false === 0);
console.log(null === 0);
```

```
false
true
false
false
false
```

!=,不等于。完全是==的相反，不要直接判断!=关系，先判断==关系，==是 true,那么!=一定是 false。同理==是 false,那么!=一定是 true。

```
// !=关系直接看==的计算结果。
console.log("12" != 12);
console.log(true != 1);
console.log(false != 0);
console.log(null != 0);
```

```
false
false
false
true
```

!==,不全等于。完全是===的相反，不要直接判断!==关系，先判断===关系，===是 true,那么!==一定是 false。同理===是 false,那么!==一定是 true。

```
// !==直接看===计算结果
console.log("12" !== 12);
console.log(12 !== 12);
console.log(true !== 1);
console.log(false !== 0);
console.log(null !== 0);
```

```
true
false
true
true
true
```

⑤NaN 比较

```
> console.log(NaN > NaN)
  false
< undefined
> console.log(NaN < NaN)
  false
< undefined
> console.log(NaN == NaN)
  false
< undefined
> console.log(NaN != NaN)
  true
< undefined
> console.log(NaN === NaN)
  false
< undefined
> console.log(NaN !== NaN)
  true
```

⑥Infinity 的比较

```
> console.log(Infinity > Infinity)
  false
< undefined
> console.log(Infinity < Infinity)
  false
< undefined
> console.log(Infinity == Infinity)
  true
< undefined
> console.log(Infinity != Infinity)
  false
< undefined
> console.log(Infinity === Infinity)
  true
< undefined
> console.log(Infinity !== Infinity)
  false
```

⑦非纯数字字符串比较。不是比较字符串的长短，而是比较字符的 Unicode 编码。顺序在靠前小于顺序靠后的。

如果第一个字符相等，依次往后进行比较。

字符的 Unicode 编码顺序：数字，大写字母，小写字母。

```
> console.log("12" < "a")
  true
< undefined
> console.log("12" < "2")
  true
< undefined
> console.log("128989898989898989" < "0")
  false
< undefined
> console.log("A" < "a")
  true
< undefined
> console.log("A" < "Y")
  true
< undefined
> console.log("banana" < "back")
  false
< undefined
> console.log("n" < "c")
  false
```

思考：3 > 2 > 1

原式= （3 > 2）> 1

= true > 1

= false

补充：控制台的快捷键

清空控制台：ctrl+L

显示上一次输入：ctrl+↑（可以无限次返回更上一次输入）

直接执行语句：enter

换行：shift+enter

1.4 逻辑运算符

逻辑运算符：

逻辑与运算： &&

逻辑或运算: ||

逻辑非运算： !

```
1
```

逻辑运算通常指的是布尔值参与运算。得到的结果也是布尔值：

1.4.1 逻辑与运算

真值表：a && b

a && b		
true	true	true
true	false	false
false	true	false
false	false	false

```
console.log(true && true);
console.log(true && false);
console.log(false && true);
console.log(false && false);
```

```
true
false
false
false
```

总结：逻辑与运算，都是 true 才是 true。有一 false 则 false。

1.4.2 逻辑或运算

真值表

a || b		
true	true	true
true	false	true
false	true	true
false	false	false

总结：逻辑或运算，有一 true，则 true。全 false 才 false.

```
console.log(true || true);
console.log(true || false);
console.log(false || true);
console.log(false || false);
```

```
true
true
true
false
```

1.4.3 逻辑非运算

非真即假，非假即真。

```
// 逻辑非运算
console.log(!true);
console.log(!false);
```

```
false
true
```

特殊情况：字符串或者数字或者特殊字符参与逻辑运算会自动转换为布尔值参与运算，得到的不一定是布尔值。

```
console.log(null || 0);
```

```
2   console.log(null && 1);
```

```
0
null
```

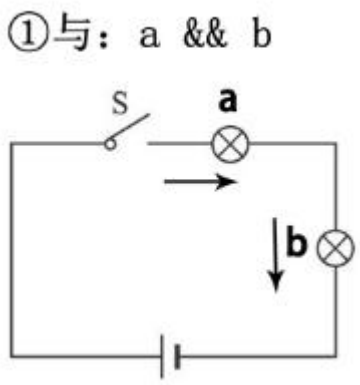
NaN，0，“”，null，undefined 会转换为 false。

非 0 的数字，非空字符串，Infinity 会转换为 true。

1.4.4 短路语法

①逻辑与&&

将逻辑与比喻成串联电路，判断过程，想象成电流通过的过程。



电流通过：如果 a 为真，电流能够通过流通到 b，结果就是 b；如果 a 为假，电流不能通过，停留在 a，结果为 a。

```
1   console.log("123" && true);
2   console.log("" && true);
3   console.log(NaN && false);
4   console.log(NaN && true);
5   console.log(0 && "1");
6   console.log(undefined && "你好");
7   console.log(null && true);
8   console.log(Infinity && 3);
```

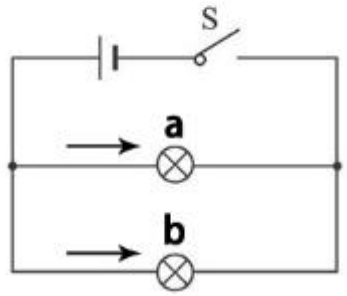
```
true

NaN
NaN
0
undefined
null
3
```

②逻辑或||

将逻辑或比喻成并联电路，判断过程，想象成电流通过的过程。

②或：a || b



电流经过时：如果 a 为真，电流直接从 a 完成循环，结果为 a；如果 a 为假，电流从 a 不能经过，流经到 b，我们结果是 b。

总结：(逻辑与 &&) 如果 a 能被转换为 false，那么返回 a；否则，返回 b。

(逻辑或 ||) 如果 a 能被转换为 true，那么返回 a；否则，返回 b。

（不要记结论，自己会推导）

```
console.log("123" || true);
console.log("" || 8);
console.log("" || null);
console.log(NaN || false);
console.log(NaN || "世界");
console.log(NaN || true);
console.log(0 || "1");
console.log(undefined || "你好");
console.log(null || true);
console.log(Infinity || 3);
```

```
123
8
null
false
世界
true
1
你好
true
Infinity
```

1.4.5 逻辑运算符的顺序

逻辑运算符的顺序：先非，与，最后或。

小测试：

```
false || !false && false || true;
```

```
原式= false || !false && false || true
    = false || true && false || true
    = false || false || true
    = false || true
    = true
```

4 && "hello" || !false || !true && null;

```
原式 = 4 && "hello" || !false || !true && null;
     = 4 && "hello" || true || false && null
     = "hello" || true || false
     = "hello" || false
     = "hello"
```

1.5 赋值运算符

一个 赋值运算符（assignment operator) 赋一个基于它右边操作数值的值给它左边的操作数。最简单的赋值运算符是等于（=），将右边的操作数值赋给左边的操作数。那么 x = y 就是将 y 的值赋给 x。

含义：将右侧的数据运算之后，赋值给左边。

在变量原基础上，进行右侧的运算，将运算的值再赋值给左边的变量。

=	直接赋值，将等号右侧值直接赋值给左侧的变量
+=	加等于
-=	减等于
*=	乘等于
/=	除等于
%=	取余等于
++	递加
--	递减

+=，加等于。相当于在原有数据之上加右侧的值运算后的结果赋值给变量（左侧的变量）

```
var a = 10;
a += 5;      //a = a + 5
console.log(a);

var b = 20;
b -= 5;      //b = b - 5
console.log(b);

var c = 5;
c *= 3;      //c = c * 3
console.log(c);

var d = 25;
d /= 5;      //d = d / 5
console.log(d);

var e = 13;
e %= 5;      //e = e % 5
console.log(e);
```

++，再原有数据之上加 1

```
var f = 10;
f ++;      //f = f + 1
console.log(f);
```

--，在原有数据之上减 1

```
var g = 10;
g --;      //g = g -1
console.log(g);
```

++，--有两种写法，有区别：

在参与运算时，写在++在前面的，先计算，计算后的值参与运算。

写在变量后的，原数据参与运算，运算之后再加 1。

（不管++在前或者在后，只要出现过一次，后面使用时都是计算后的一个值）。

```
var a = 10;
var b = 20;
var c = 30;
// 10 21 30 12
var d = (a ++) + (++ b) + (c ++) + (++ a);
console.log(d);
```

1.6 综合运算

顺序：贴身的（! ++ --）→ 数学 → 比较 → 逻辑 → 赋值

计算：

var a = 4;

var sum = 1 * (2 + 3) && a++ || 5 > 6 && 7 < 8 || 9;

```
原式 = 1 * (2 + 3) && a++ || 5 > 6 && 7 < 8 || 9
     = 1 * (2 + 3) && 4 || 5 > 6 && 7 < 8 || 9
        = 5 && 4 || 5 > 6 && 7 < 8 || 9
        = 5 && 4 || false && true || 9
        = 4 || false || 9
        = 4 || 9
        = 4
```

```
4
```

练习：

```
var a = 4;
1 + 2 && 3 * a++ % 5 || 6 < 7 == 8 / !false
```

```
        // 原式 = 1 + 2 && 3 * a++ % 5 || 6 < 7 == 8 / !false
        //      = 3 && 2 || 6 < 7 == 8
        //      = 3 && 2 || true == 8
        //      = 3 && 2 || false
        //      = 2 || false
        //      = 2
```

```
2
```

二、流程控制语句

我们的 js 通常执行顺序，是从上到下依次。

但是我们可以使用其他方法控制 js 的执行顺序。也就是控制一些语句先执行或者控制一些语句不执行。

常用的是条件分支语句：

分类：if 语句、三元表达式、switch 语句。

2.1 if语句

2.1.1 简单的 if 语句

if....else...

if：如果

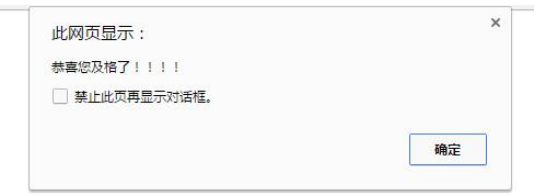
else：否则

抽象模型：

```
if(条件表达式){
    当条件表达式为真时，执行的结构体。
}else{
    当条件表达式为假时，执行的结构体。
}
```

结构体：就是多个语句的一个集合。

```
var a = parseInt(prompt("请输入您这次考试的成绩"));
        if(a >= 90){
            alert("恭喜您及格了！！！！");
            alert("别骄傲，继续努力");
        }else{
            alert("虽然没有及格但是下次继续努力");
        }
```



if 语句可以不写 else。当 if 条件表达式为假时，直接跳出 if 语句，执行 if 语句后的其他语句。

```
var a = parseInt(prompt("请输入您这次考试的成绩"));
        if(a >= 90){
            alert("恭喜您及格了！！！！");
            alert("别骄傲，继续努力");
        }
        alert("谢谢您参与这次考试！");
```

当结构体只有一句 js 语句时，可以省略大括号（但是不推荐使用）

```
var a = parseInt(prompt("请输入您这次考试的成绩"));
        if(a >= 90)
            alert("恭喜您及格了！！！！");
        else
            alert("虽然没有及格但是下次继续努力");

        alert("谢谢您参与这次考试！");
```

2.1.2 多分支 if 语句

可以写多个 else if(只能出现一次 if 只能出现一次 else,可以出现多次 else if)。

```
if(条件表达式 1){
    满足表达式 1，执行的结构体
}else if(条件表达式 2){
    不满足表达式 1，满足表示式 2 执行的结构体
}else if(条件表达式 3){
```

```
6       不满足表达式 1 和表达式 2，满足表达式 3 执行的结构体
7   }else{
8       都不满足执行的结构体。
9   }
```

我们将考试成绩分为了 4 个档次，一个成绩只能出现一个档次内。

```
1   var a = parseInt(prompt("请输入您这次考试的成绩"));
2   if(a >= 90){
3       alert("优秀");
4   }else if(a >= 80){//不用写 80 <= a < 90 ,含义直接就是不满足第一个条件。
5       alert("良好");
6   }else if(a >= 60){
7       alert("及格");
8   }else{
9       alert("同学真该努力了");
10  }
```

“跳楼现象”，多个 else if 语句满足跳楼现象。也就是结构体只能执行一次。满足某个条件表达式时意味着他不满足前面所有的表达式。也就是说一个 if 语句只能执行一次。

也可以不写 else，当都不满足条件时，直接跳出 if 语句之后后面的代码。

```
1   var a = parseInt(prompt("请输入一个 10 以内的整数"));
2   if(a <= 3){
3       console.log(a += 2);
4   }else if(a <= 5){
5       console.log(a += 2);
6   }else if (a <= 8){
7       console.log(a += 3);
8   }else{
9       console.log(a ++);
10  }
```

当输入一个 2 时，满足 a<=3，直接输出 4，不会再往下执行。

当输入一个 8 时，不满足第一层，直接进入第二层（a<=5）。也不满足条件，再进入第三层，满足 a<=8 直接执行结构体。输出 11。不在往下继续执行。

if 语句嵌套

if 语句里面可以放多个 if 语句：

```
1   var sex = prompt("请输入您的性别");
2   var age = parseInt(prompt("请输入您的年龄"));
3   if(sex == "男"){
4       if(age >= 22){
5           alert("小伙你可以结婚了");
6       }else{
7           alert("再等等吧");
8       }
9   }else{
10      if(age >= 20){
11          alert("小姑娘你可以结婚了");
12      }else{
13          alert("小姑娘在等等吧");
14      }
15  }
```

```
1
```

```
1
```

```
1
```

```
1
```

```
1
```

```
1
```

```
1
```

```
1
```

```
1
```

```
1
```

```
1
```